An MPEG-oriented platform for Wave Field Synthesis arrays of loudspeakers

Giorgio Zoia, and Claudio Alberti
Integrated Systems Laboratory, EPFL
Lausanne, CH-1015, Switzerland

The MPEG-4 standard provides a complete framework for hybrid coding of natural and structured sound information that permits the description of complete spatial environments by a low amount of data. A new platform, based on a virtual DSP, has been developed to optimize the MPEG decoding and pre-processing interface to feed an array of loudspeakers rendering spatial Audio by Wave Field Synthesis. The virtual DSP has an instruction set adapted to MPEG-4 advanced audio features on superscalar processors.

WAVE FIELD SYNTHESIS AND MPEG-4

In the late eighties a fundamentally new approach to multichannel sound rendering was proposed, namely Wave Field Synthesis (WFS) [1]. With this new approach, wave theory plays a fundamental role and individual loudspeakers are replaced by arrays of loudspeakers that generate wave fronts from true or notional sources [2]. Unlike almost all the other existing methods, the wave front solution can be called a volume solution, i.e. it generates an accurate and more realistic representation of the original wave field in the entire listening space.

In the ideal case, the listening area is surrounded by planes of loudspeakers, which are fed with signals so that they produce a volume flux proportional to the normal component of the original sound field at the corresponding position [3]. For more practical purposes, the method has been adapted to make use of linear arrays of loudspeakers, which surround the listening area, in substitution of the planes of loudspeakers [3],[4].

Despite of this remarkable simplification, the WFS approach is still characterized by a huge number of output channels (typically more than 100), necessary to generate the sound field. This makes the storage and/or transmission of data, uncompressed or even coded, an important problem to be solved. In this context, it is fundamental to be able to deal with the sound in terms of a mixed recorded/structured description [5]; as a consequence, a considerable amount of processing power is required in the rendering device terminal to reconstruct a recorded (or even synthetic) acoustic environment starting from dry, monophonic sources.

Current implementations of spatial rendering schemes are based on synthetic models of room impulse responses, containing delay lines for the simulation of early reflections, plus a module for the generation of a diffused late reverberation. The model-based approach can also be applied to Wave Field Synthesis [4]; the main advantage of a model-based system is scalability, i.e. the complexity of the model can be scaled according to the available processing power; moreover, in a virtual reality interactive context, the parameters can be recalculated with moderate processing cost when source positions are changed in real-time.

On the other hand, in a data-based scheme each source signal is convoluted with long FIR filters, as many filters as the number of channels of the speakers array [6]. The FIR filter coefficients are directly computed from measured or simulated room impulse responses. A very high degree of realism can be achieved, if, for example, the environment that should be reproduced is measured by a suitable microphone array [7].

At this point, it is important to investigate the availability of a standardized audio description format, able to support and encode in an appropriate way the information required by a WFS-based system.

Advanced Audio Functionality in MPEG-4

The new MPEG-4 Audio standard provides two toolsets to code advanced multimedia-oriented audio content, namely Structured Audio (SA) and BInary Format for Scenes (BIFS).

SA derives from academic research in software sound synthesis (SWSS) languages [8]; the SAOL (Structured Audio Orchestra Language, see [9]) programming language is similar to many of these SWSS tools, but the SA toolset as a whole has been conceived for multimedia and downloadable applications; the syntax of the language itself has become very close to that of the C language, with stronger object-oriented connotations and several stream-oriented extensions.

SAOL maintains the typical execution scheme of its predecessors, with an internal scheduler (the fixed "main" of the program) and the possibility to define, in its synthesis and processing functions, variables with different execution rates: initialization, control and audio rate. But unlike other similar tools, characterized by a block-based syntax, SA has a sample-by-sample (s-b-s) execution structure: this means that syntax and semantics of statements and operators are defined for a single sample and not for a block of samples of length B_l

$$B_l = \text{srate/krate} \qquad (1)$$

where srate and krate are the sampling rate and the control rate respectively. If this makes possible a correct implementation of basic functions like recursive filters, on the other hand it introduces a relevant overhead in the case of an interpreted implementation of the language decoder, the most suitable for embedded real-time engines. Note that a structured audio coding scheme can be seen as a general coding approach [10]; in these terms, decoding is used as a synonym of program execution.

In MPEG-4, SA is surrounded at the system layer by a mark-up language for scene description, BIFS; it is a language very similar in its hierarchical structure to VRML, but with an innovative audio-specific subtree called AudioBIFS [10]; functionality of AudioBIFS nodes ranges from simple mixing or delay until advanced spatialization schemes based on geometrical and perceptual information, so that a complete virtual audio environment can be described. Custom processing algorithms, described by means of specific SAOL code, can be inserted in the scene tree through the AudioFX node, making of AudioBIFS an extremely powerful and flexible environment. An example of an AudioBIFS subtree is represented in Figure 1.

MPEG-4 for Wave Field Synthesis

In several ways MPEG-4 AudioBIFS corresponds to the requirements of the powerful WFS system, since it allows to code most of the necessary information in a high-level, structured format. As already remarked earlier, this permits to avoid waste of bandwidth or storage space but, of course, it implies a more demanding effort at the decoder side, where the rendering device needs to compute the several audio channels using one of the two models presented above.

Once the dry monophonic sources are properly coded by high-quality tools also provided in MPEG-4, there are two possibilities to code by the AudioBIFS format the high-level side information for WFS, corresponding to each one of the two approaches. In a first way, that is suitable for the more realistic data-based approach, it is possible to transmit the whole data, FIR coefficients or room responses, once over the channel; this can be done for instance including responses as tables of an SA object. Once forwarded to the filters, these data can be deleted from the decoder memory by MPEG-4 stream commands and only coded monophonic sources are further received.

Otherwise data can be computed out of the existing geometric or perceptual environmental descriptions that are supplied by MPEG-4 AudioBIFS (this method is suitable for the model-based approach, and it also needs to be implemented to guarantee compliance with the standard). Aiming at a practical system implementation, the data-based approach is more convenient for the several reasons explained in [6]; in this case the BIFS interpreter must pre-process the control information and supply to the WFS devices the same type of information that is made available by the data-based approach.

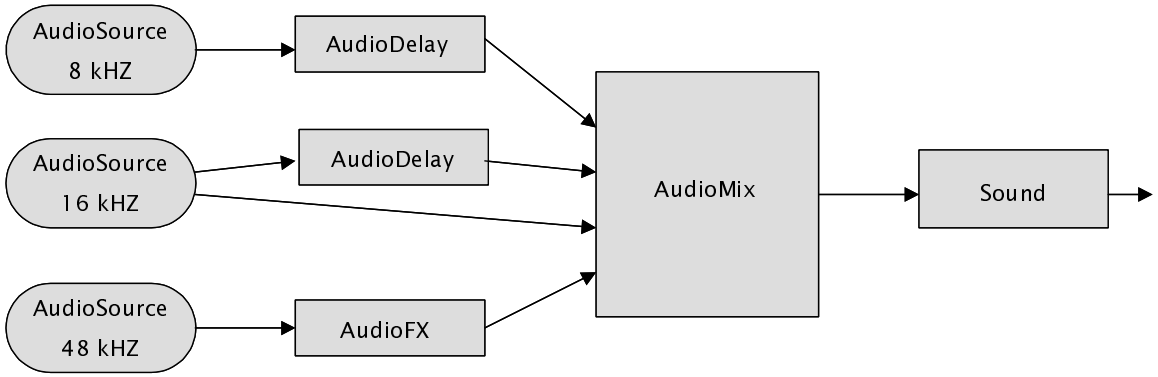

Figure 1 Example of Audio subtree in MPEG-4

AN MPEG-ORIENTED VIRTUAL DSP

The SAINT (for Structured Audio INTerpreter) SA decoder finds its foundation in a systematic approach to the estimation of the SA decoding complexity, which must necessarily move from a time-dependent analysis and profiling of its typical applications. A new abstract method for measuring decoding complexity of MPEG-4 Structured Audio programs has been developed [12] that provides platform independent metrics and that permits to carefully profile the execution of a program in function of time; in this way it is possible to characterize critical situations and to define a subset of the SAOL standard core library that is candidate for stronger optimization [13].

Feedback analysis

In parallel with the analysis of complexity, a second fundamental decoding issue has been considered: the possibility of a block-by-block (b-b-b) execution for SA, without altering the output of the normative s-b-s language specification.

Efficiency of a block-based execution over a sample based one has been previously proofed in literature [14]. In SA, what can prevent from executing b-b-b is the presence of an explicit feedback in the SAOL code. By explicit feedback it is intended here a feedback programmed using more than one line of code, while an implicit one is for instance the case of the iir library function, where the feedback is hidden at a lower level, visible by the interpreter but not by the programmer. Explicit feedbacks have been detected by a simple graph analysis only in a few situations. The most obvious is when a new value is assigned to an audio variable after its first use. These cases have to be detected and treated in a special way, while the rest of the code can be executed on a possibly large b-b-b basis, typically with a block length of B_l as defined in (1). Of course, this can be done until the latency introduced by the block processing in the real-time synchronization of the complete MPEG-4 decoding process is tolerable.

The virtual DSP architecture

Combined analyses of complexity and feedback demonstrated that in most cases an efficient implementation of the SA decoder can be obtained by the design of an extended multimedia DSP, based on a vectorial instruction set [13]. Many multimedia-oriented superscalar devices are nowadays available, and some of them specifically designed to support multichannel and spatial sound processing. This is the main reason behind the choice to design a virtual device with an instruction set that well matches the parallelism exploitable in these state-of-the-art DSPs and processors [15].

With the SAINT approach, the entire decoding is split between two independent layers: the decoder/compiler layer and the instruction layer. The complete process is divided into two separable parts, the initialization task (related to the structured description of sound) and the processing task; in this way, it is not difficult to run the first phase on a general purpose processor, and to execute the intensive processing possibly in the same CPU, but with the same effectiveness in a separate co-processor, single or even distributed. This structure is conceived to target embedded solutions for standalone devices, typical in the case of high-quality multichannel Audio systems. In fact, they usually contain custom solutions based on superscalar processors and DSPs.

The SAOL compiler, entirely written in C for portability, is completed by a post-processor that optimizes some parts of the code and performs the feedback detection analysis over the potential cases.

The virtual DSP engine

The instruction set of the SAINT virtual DSP is essentially composed by macroinstructions, so called because they are conceived to operate on vectors of samples. These macroinstructions are the methods that are directly executed by the arithmetic unit (ALU) of the machine. All of them are defined in vectorial form, where the block of data on which the instruction is executed is defined by the values of two special registers in the machine, *start* and *end*. In SAINT there is no memory stack to work on, all the instructions directly operate on memory locations, and then they are defined with two or three addresses to load data and store results.

The scheduler works in close conjunction with the ALU of the DSP, and then it must run on the same device where the code is being executed, to avoid useless communication delays between the two units. In fact, the scheduler can be seen as a master DSP unit working together with the processing DSP and executing a fixed, "hardwired" program of control and data post processing.

The virtual DSP architecture has been tested by several measurements with different versions of the decoder. Experiments on performance have shown an improvement of approximately a factor 20 in speed over the MPEG-4 reference software and a slowdown of 20-30% in comparison with a cross-compiled SAOL-to-C decoder [13].

JOINING MPEG-4 AND WAVE FIELD SYNTHESIS TOGETHER

An architecture and an instruction set conceived and optimized only for Structured Audio reveal several limitations when the device must be used for more general multimedia-oriented applications. Even if SAOL can potentially be used to describe any kind of algorithm, being a programming language and then a general audio coding scheme [10], many fundamental features of typical advanced audio applications, such as synthetic reverberation for multichannel audio, cannot be described with the necessary effectiveness.
Moving from the purpose of extending the MPEG-4 decoder to a complete AudioBIFS subtree processor, several extensions have been included in SAINT to support at best functionality of the BIFS language and to approach the requirements to interface the WFS final rendering stage.

Extensions for Virtual Audio Scenes

The implementation of the complete AudioBIFS subtree requires the solution of non-trivial implementation challenges, first of all the correct synchronization between the Structured Audio built-in scheduler, which is active to process the AudioFX nodes, and the BIFS scene scheduler. An evaluation of practical implementation issues suggests that an efficient solution can be the integration of the complete audio subtree into an extended *orchestra* (the collection of functions to run an SA program), where all the nodes are transformed into SAOL algorithms and linked to the input/output buffers of the eventual AudioFX nodes in the correct sequence. It was then decided to "cross-compile" the AudioBIFS subtree to a sequence of extended SAOL functions and to proceed then to the execution of the complete audio scene using the SA scheduler as the master mechanism for synchronization.

We mentioned earlier the necessity to extend the SAOL functionality; these extensions are necessary on both the instruction set and the memory management of the virtual DSP, to deal essentially with the more interactive nature of BIFS.

In SAOL the several instruments (i.e. the synthesis or processing functions) are statically defined at the beginning of the decoding process, when the bitstream header is received, and so are the routings among them; routings define the relationships among the input and output buffers of the different instruments. Only new instances of an existing instrument can be transmitted by streaming information.

In BIFS this is not enough anymore, because routings among the nodes are dynamic and new nodes can be instantiated in the middle of the scene representation, when new commands are received via the bitstream. Since in the proposed system the interaction between the parser/compiler and the execution engine is kept as far as possible separate, the virtual DSP must be made able to process streaming commands dealing with dynamic configurations of the input/output buffers, and above all with creation of new instruments. For instance, a new mixing AudioMix node could be instantiated, with a mixing matrix processing four inputs to produce two outputs, linked with sources and to target post-processing nodes.

A second class of extensions is necessary to support functionality provided by the more advanced nodes of BIFS. If some functions like mixers, delay lines and switchers are not difficult to translate into SAOL, on the other hand AudioBIFS is suitable for a multichannel WFS rendering system because it provides the possibility to spatialize monophonic sources using schemes such as room modeling or perceptual parameters. It is evident that in such cases some dedicated instructions are necessary to limit the overhead of the interpreted decoder to a minimum and to better optimize the transcoding into the intermediate format.

Finally, BIFS allows the presence of nodes working at different sampling rates, and lower rates must be converted to the higher ones at the most advanced point in the processing tree [16]. This is not possible in standard SAOL, where all the instruments must work at the same audio rate. It is then necessary to extend the scheduler with the capability to treat different subgroups of instruments (i.e. different orchestras) that are sequenced through asynchronous sampling-rate converters at the correct point.

ThreeDSPACE: MPEG-4 driving Wave Field Synthesis

The concept and design of SAINT and of its extended BIFS version are inserted in the framework of ThreeDSPACE. ThreeDSPACE is a mixed academic/industrial research project aiming at implementing advanced 3-D audio processing by Wave Field Synthesis. In it, SAINT is used as signal generator and preprocessor, producing the signals and required side controls as described in the first section. The composition of the AudioBIFS scene is not completed by the virtual DSP itself. Instead, monophonic audio sources and side information for spatial control are sent to the output of a first processing stage, and another active stage driving loudspeaker stripes is in charge to produce the

necessary number of channels (up to 64 per stripe) in order to generate the holophonic sound field. The overall system architecture and related data flow used in ThreeDSPACE is represented in Figure 2.

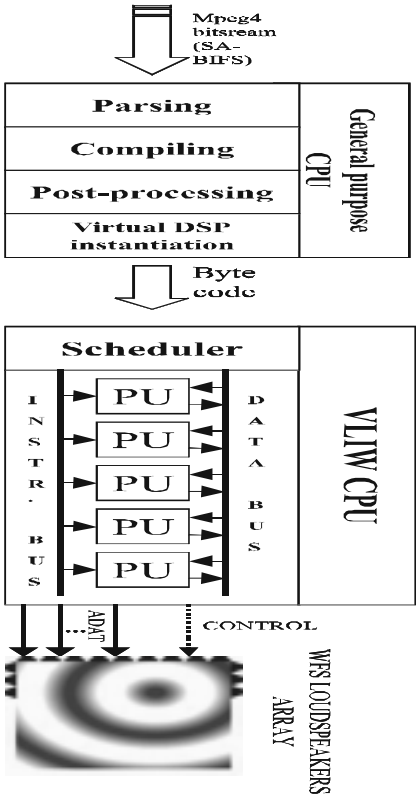

Figure 2 The data flow through the ThreeDSPACE audio framework. Mono channels and control information are processed by active arrays of loudspeakers to produce suitable WFS multiple channels

The entire system can be split into three main stages. The first one, dealing with the parsing and compilation of the MPEG-4 bitstream, has been implemented on a general purpose CPU platform. The second stage supports the virtual DSP and has been ported on a TriMedia multimedia DSP, a V-LIW (Very Long Instruction Word) architecture. In the first version of the system, this processor runs at 100MHz; its V-LIW instruction set allows up to five simultaneous operations to be issued. The proposed block-by-block approach shows its potential exploiting this feature, reaching average values of 4.5 processing units busy per clock cycle, as it will be shown in more details in the last section. The aggressive exploitation of parallel features lets the virtual DSP run at a speed greater than the one obtained on a 200MHz general purpose Pentium CPU, with output audio formatting in addition.

Monophonic output channels are sent in ADAT format to the last processing stage, where convolutions are executed to implement the data-based approach to Wave Field Synthesis. A detailed description of this last stage can be found in [6]. A single ADAT link is used; seven monophonic sources at 48 kHz can theoretically be carried on the first seven channels, while the eighth channel is dedicated to control information, which is transmitted according to a synchronous specific protocol. A graphical representation of this link is reproduced in Figure 3.

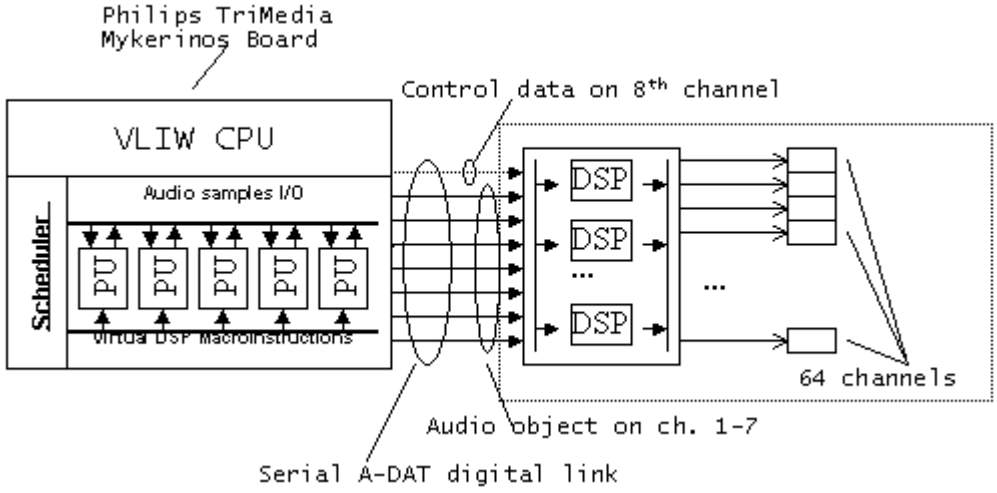

Figure 3 Block-diagram of the ThreeDSPACE rendering system. A one-directional optical link is enough to guarantee the necessary information flow from the BIFSAINT execution engine to the active arrays of loudspeakers.

PORTING SAINT ON A VLIW ARCHITECTURE

One of the fundamental issues on which the whole SAINT architecture has been conceived is the possibility to implement and optimize it at a reasonable cost on superscalar DSP architectures, with the concrete possibility to actually exploit the parallelism and the eventual advanced audio support they provide. The porting of the SAINT virtual DSP to the VLIW TriMedia processor has represented an important test bench for the SAINT concept and for the ThreeDSPACE system as a whole.

The heart of TriMedia is its 32-bit DSPCPU core. The DSPCPU implements a 32-bit address space and 128 general-purpose 32-bit registers. The registers are not separated into banks; any operation can use any register for any operand. The core implements a VLIW architecture that provides up to five simultaneous operations to be issued. These operations can target any 5 of the 27 functional units in the DSPCPU. These units include integer and floating-point arithmetic units and data-parallel DSP-like units. SIMD (Single Instruction Multiple Data) operations are possible on 8-bit or 16-bit integer data. Being the floating-point format the standard required MPEG-4 SA format, this SIMD add-on and five integer units have little impact on the considered application. Two different caches are also present: 32 kB for instructions and 16 kB for data.

Code Optimization

Assembly language for the VLIW instruction set is too complicate to be well managed by humans and then it is understandable how fundamental is a robust and optimized compiler.

Given the architecture of SAINT, it makes sense to try first of all to optimize the implementation of the vectorial instruction set, being the scheduler a separate and low cost task, which moreover does not offer many indications for a data-level parallelism. It is interesting indeed to report the work of optimization of the vectorial instruction set; this task corresponds to the implementation of a vectorial DSP library for the VLIW CPU, and it is of general interest for many kind of parallel DSP processing, including of course channel generations for WFS.

Compilers for superscalar architectures normally provide some dedicated optimization facilities intended to better exploit the parallelism of the specific device. Often these techniques come with some practical examples that can be useful to produce code more suitable for aggressive parallelization [17]. At the same time programmers know techniques that can be used to reduce overhead in a code, like for instance loop unrolling or manipulation of pointers.

Together with all these potential optimization techniques, special pointers can be declared, called restricted pointers; it is interesting to spend some words on them, since experimental results reveal how only their use can produce real parallel execution. At the compilation time, the compiler does not know if a pointer to an allocated memory area is overlapped with another pointer possibly accessing the same data to modify them in a read/write process. That is to say, the compiler normally assumes that two pointers may refer to the same or overlapping memory locations, i.e. be aliased to each other. This implies that operations of two different statements having those pointers as operands cannot be executed in parallel. However, if it is known that all the pointers point to distinct memory areas and thus never alias, it is possible to convey this information to the compiler by declaring these pointers as restricted. Based on this information, the compiler decides that different variables and/or restricted pointers do not alias. It is a programmer's responsibility to verify that the assertion is true; proper use of restricted pointers reduces the amount of dependencies and, therefore, increases dramatically potential parallelism (see experimental results later).

In the case of SAINT, and its BIFS extension in general, the vectorial macroinstruction intrinsically contains the certainty that memory areas are not overlapped. In fact, detection of feedback is performed by the SAOL compiler before execution of the bytecode, and then potential overlappings are already eliminated and relating instructions treated as sample-by-sample (that again avoid overlapping by making the virtual DSP work on one sample at a time). Concerning instead the cross-compilation of AudioBIFS to SAOL, thing that is more specifically used with WFS to pre-process structured information and generate driving controls, the resulting SAOL code is statically optimized in advance and some reconfigurable templates are used, thing that again assures the presence of a one-directional data flow inside the node processing.

Experimental Results

In this subsection experimental results are reported that show how the vectorial instruction-set matches the parallel potential of the V-LIW processor and the multichannel pre-processing can greatly benefit from this particular structure. Results are shown in Figure 4 and Figure 5 for a 4^{th} order IIR filter; given the structure of the code, these results do not change meaningfully for an 8^{th} order FIR filter. In

ThreeDSPACE convolutions are performed in the last processing stage, and then on a different platform, but this benchmark is anyway meaningful as it represents a large class of operations that are normally executed during the BIFS subtree processing. Moreover, it also shows how the last WFS stage could also benefit from such architecture.

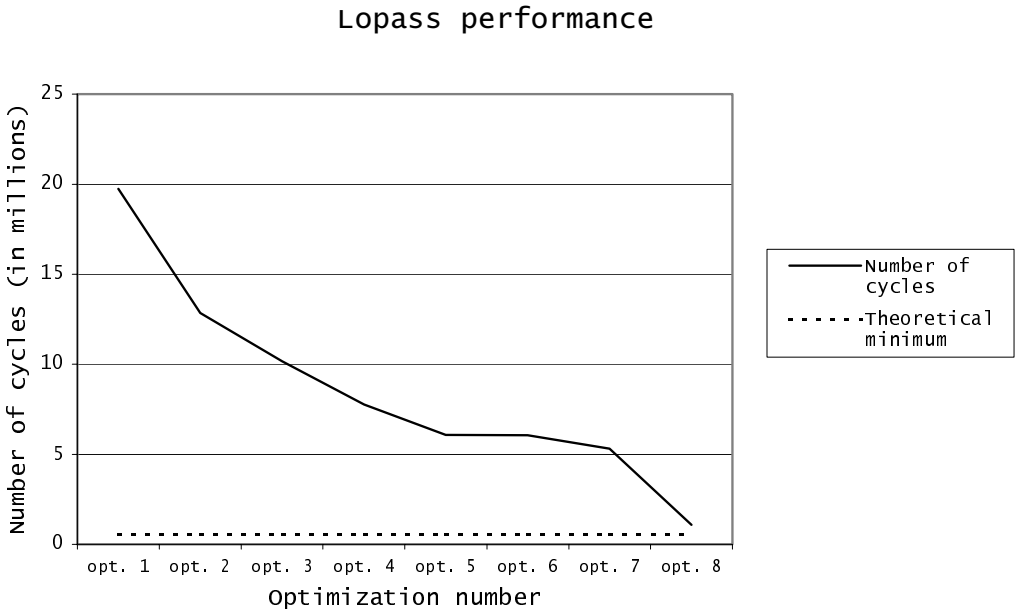

Figure 4 Performance of the FIR/IIR filter on a large block of samples for different levels of optimization

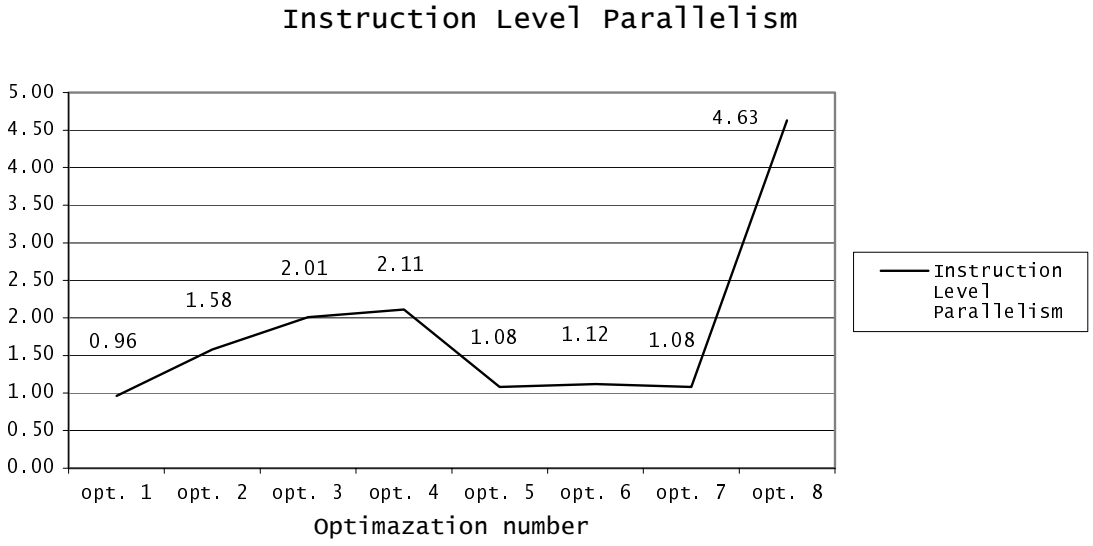

Figure 5 Instruction Level Parallelism (ILP) of the FIR/IIR filter on a large block of samples for different levels of optimization

The experiments were made on a large block of samples, to isolate real computation and reduce potential overhead coming from too many cache memory refreshes. The vertical axis in Figure 4 represents the million of cycles to filter $2*10^5$ samples, while the horizontal axis reports the described methods of optimization in the following order:

Opt.1 Profile-Drivel compilation
Opt 2 Decision-tree grafting
Opt 3 Loop optimization
Opt 4 Loop optimization: local reference parameters
Opt 5 Loop fusion
Opt 6 Manual Loop unrolling
Opt 7 Manipulation of pointers
Opt 8 Restricted pointers

The first four types of optimization are automatically inserted by the superscalar compiler, while optimizations from 5 to 7 are well known to parallel programmers [18]. Optimization 8 has been extensively described above. In the two Figures, 4 and 5, optimizations are cumulated, i.e. in column 2 optimizations 1 and 2 are used and so on. The theoretical minimum in Figure 4 is calculated in a conservative way, counting only multiplications and additions; considering that the TriMedia cannot allocate the five slots to floating-point operations, it is supposed that at the same time necessary loads and stores can be performed. A theoretical minimum is often difficult to estimate on a parallel architecture and then this comparison must be taken as a not too strict reference, with some margin of error.

On the other side the values reported in Figure 5 are objective, since the vertical axis represents the calculated Instruction Level Parallelism (ILP), which for the considered device has a precise upper limit of 5 (no more than five execution slots can be allocated per cycle). In this case the open issue could be if all the operations executed in parallel are necessary or not, but the theoretical limit is precise. Given these two groups of measurements and the potential margins of error they contain when considered separately, conclusions have to be drawn by a comparative analysis of the two figures.

The first four optimizations introduce a reduction of about 2.3 times in the number of cycles. This has its counterpart in an increased ILP from 0.96 to 2.11 and then it is evident that there is no reduction in the overall number of executed operations but only the compiler is able to detect some code inefficiencies and to translate them into parallel operations. When some handcrafted code manipulation is introduced, loop fusion and manual unrolling, the number of cycles further reduces but the ILP falls down to nearly 1. This has a simple explanation: the parallelism introduced by the compiler so far was a false one, in the sense that it compensated code written in a too formal way: once these formal redundancies are removed it results clear that only one multiplication or one addition are executed at a time and no true parallelism is exploited at all. Change in the way of addressing arrays does not improve the situation, it just reduces a little the number of cycles.

The real improvement comes at the last step, where the structure of the instruction set of SAINT and its execution procedure allow the indiscriminate application of restricted pointers. This special declaration tells the compiler that it can really schedule processing to happen in parallel, and while the ILP jumps by a factor of almost 4.3 at the same time the number of cycles decreases of a factor 4; this means that in this case the processor is really capable to calculate two floating point operations and two loads or stores in parallel, also being capable at the same time to manage the integer arithmetic of the outer for loop from start to end.

A DSP V-LIW vectorial library

Needless to say that not for all the macroinstructions of SAINT it was possible to reach a performance near to the theoretical limit and with high degrees of parallelism.

In particular operations containing calls to precompiled mathematical functions constitute a delicate group. It is evident that without a complete vectorial set of operations it is not possible to really speed-up the performance of the SAINT engine on the whole range of functionality.
On the other side, writing vectorial libraries for mathematical and trigonometric operators on a truly parallel processor requires a great skill in numerical computation techniques and could constitute a research topic in itself. Being the purpose of this relevant optimization effort to build an efficient virtual DSP execution engine for the MPEG-4 AudioBIFS subtree and for WFS control in particular, many specific SA functions in strong relation with Audio synthesis has not been considered yet.

CONCLUSION

We have presented in this paper the design and implementation of a virtual DSP platform aiming at the realization of a multichannel reproduction system based on Wave Field Synthesis. The virtual DSP is

conceived to decode the MPEG-4 AudioBIFS subtree (including Structured Audio) with a great efficiency on superscalar devices.
The architecture of the complete WFS-based rendering system has also been presented; it is build on a three-stage processing chain obtained by carefully splitting general-purpose and highly dedicated processing tasks.

We would like to thank dr. Ulrich Horbach and Attila Karamustafaoglu for helping us in understanding many fundamental issues that made our work fruitful. Many thanks to Daniel Farre for his precious help.

REFERENCES

[1] A.J. Berkhout (1988). *A holographic approach to acoustic control*. Journal of the Audio Engineering Society, vol. 36, pages 977-995.

[2] U. Horbach and M. Boone (1999) *Future Transmission and Rendering Formats for Multichannel Sound*. Proceedings of the AES 16th International Conference on Spatial Sound Reproduction, Rovaniemi, Finland.

[3] A.J.Berkhout, D. de Vries, P. Vogel (1993) *Acoustic Control by Wave Field Synthesis*. Journal of the Acoustic Society, vol. 93, pages 2764-2778.

[4] M. Boone, E. Verheijen, G. Jansen (1993) *Virtual Reality by Sound Reproduction Based on Wave Field Synthesis*. Proceedings of the 100th AES Convention, Copenhagen, Denmark.

[5] G. E. Garnet (1991) *Music, Signals and Representations: A Survey*. In Representations of Musical Signals, G. De Poli, A. Piccialli, C. Roads Editors, pages 325-370, MIT Press, Cambridge, U.S.A.

[6] U. Horbach, A. Karamustafaoglu and M. Boone (2000) *Practical Implementation of a Data-Based Wave Field Reproduction System*. Proceedings of the 108th AES Convention, Paris, France.

[7] W. de Bruijin, T. Piccolo and M. Boone (1998) *Sound Recording Techniques for Wave Field Synthesis and other Multichannel Sound Systems*. Proceedings of the 104th AES Convention, Munich, Germany.

[8] C. Roads (1996) *The computer music tutorial. Part II, Sound Synthesis.* MIT Press, Cambridge, U.S.A.

[9] B. Vercoe and E. Scheirer (1999) *SAOL: the MPEG-4 Structured Audio Orchestra Language*. Computer Music Journal 23 (2) pp. 23-35.

[10] E. Scheirer and Y. Kim (1999) *Generalized Audio Coding with MPEG-4 Structured Audio.* Proceedings of the AES 17th International Conference on High-Quality Audio Coding, Florence, Italy.

[11] E. Scheirer, R. Väänänen and J. Huopaniemi *AudioBIFS: Describing audio Scenes with the MPEG-4 Multimedia Standard.* IEEE Transactions on Multimedia, vol. 1, no. 3, pages 237- 250.

[12] G. Zoia (1998) *A method for Complexity Measurements in Structured Audio*. ISO/IEC JTC1/SC29/WG11 (MPEG98) contribution document M3602.

[13] G. Zoia and C. Alberti (2000) *A Virtual DSP Architecture for MPEG-4 Structured Audio.* Proceedings of the COST-G6 Conference on Digital Audio Effects - DAFX'00, Verona, Italy.

[14] R. Dannenberg and N. Thompson (1997) *Real-Time Software Synthesis on Superscalar Architectures*. Computer Music Journal, vol. 21, no.3, pages 83-94.

[15] R. Espasa and M. Valero (1997) *Exploiting Instruction- and Data-Level Parallelism*. IEEE Micro, vol. 17, no. 5, pages 20-27.

[16] ISO/IEC JTC1/SC29/WG11 (1999) *Information Technology - Coding of Audio-Visual objects. Part 1: Systems*. MPEG-4 Systems International Standard.

[17] G. Slavenburg et al. (1998) *TM1100 Preliminary Data Book*. Philips Electronics NAC.

[18] D. Skillicorn (1994) *Foundations of Parallel Programming*. Cambridge University Press, Cambridge, U.K.